システムソフトウェアとオペレーティング・システム 86－5
(2001. 3. 5)

リアルタイムモニタリング機能を実装したハードウェアスケジューラの設計と実装

河村 透　中西 恒夫　最所 圭三　福田 晃
奈良先端科学技術大学院大学 情報科学研究科
〒 630-0101 奈良県生駒市高山町 8916-5
{toru-ka, tun, fukuda}@is.aist-nara.ac.jp,　sai@eng.kagawa-u.ac.jp

あらまし

ハードリアルタイムシステムを実現する上で、制限時間の厳守や応答時間の短縮を図るため, OS の機能の一部であるスケジューラをハードウェアで構成したハードウェアスケジューラと、プロセッサの実行状況を監視するリアルタイムモニタリング回路を開発し、プロセッサの実行状況に応じたスケジューリングができるアーキテクチャを提案する。その結果ハードウェアスケジューラは汎用プロセッサよりも短いステップでスケジューリングを行えることを確認した。

キーワード　ハードリアルタイムシステム, スケジューリング, **EDF** アルゴリズム, **FPGA**, モニタリング

Design and Implementation of a Hardware Scheduler with Real-Time Monitoring Function

Toru Kawamura　Tsuneo Nakanishi　Keizo Saisho　Akira Fukuda
Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0101, Japan

Abstract

We construct a hardware scheduler which monitors the condition of a running processor in real-time. The purpose of the scheduler is to keep the hard deadline strictly and to shorten response time. The hardware scheduler, which replaces the software scheduler built in the real-time operating system, traces status of real-time task changing frequently and performs robust real-time scheduling.

Our results show that the hardware scheduler reduces the steps which requires to schedule tasks.

key words　hard real-time system, scheduling, EDF algorithm , FPGA, monitoring

1 はじめに

組込みシステムのアプリケーションは様々あるが、中には制限時間内に処理が終了しなければならないリアルタイム性を要求されるものが多くある。このようなシステムでは決められた制限時間までに所定の処理が完了することが保証されてなければならない。この時間制限(デッドライン)を特に重要視するシステムをハードリアルタイムシステムという。このシステムを構築するためにはプログラムの実行を管理して制限時間内に終了させる機構が必要である。リアルタイムOSではスケジューラがこの役割を果たす。スケジューラはタスクに割り振られたプライオリティに従って、タスクの実行順序を決める。ハードリアルタイム処理を実現するスケジューラは更に最悪実行時間とデッドラインを考慮してスケジューリングを行う。

ハードリアルタイム処理を実現するスケジューリング法の一つであるEDF(Earliest Deadline First)スケジューリング法は最大遅れ時間を最小化する最適化スケジュール法であるが、プロセッサへの負荷が他のアルゴリズムと比較して最も高いため、使用できる機器が限られる欠点がある。またEDFアルゴリズムはオーバーロード状態に非常に弱く、次々と新たなタスクが起動されることで、それまでスケジューリングの結果デッドラインを守れるはずだったタスクが次々とデッドラインを守れなくなる“将棋倒し現象”を最も起こしやすい。

更に割り込み処理における割り込み応答時間をスケジューラは把握できないため、スケジューリング結果に不一致が生じる可能性がある。割り込み応答時間とは割り込みが生じてから割り込みハンドラが割り込み処理を行なうまでの時間である。これら一連の割込み処理は例外的なものであるため、スケジューラは割込みハンドラの操作、実行時間も含め、この割込み処理を関知していない。そのため割込み処理から元のタスクの処理に戻ってきたときに、既にタスクはデッドラインを超過している可能性がある。割込み応答時間をスケジューラが把握できないために生じるこの状況はハードリアルタイムシステムにとって危惧すべき要因となる。

そこでプロセッサの負荷の軽減やオーバーロード時の対処、OS設計の簡素化を図るべく、スケジューリング専用に作られたLSIを用いてハードリアルタイム処理を実現するシステムを提案する。このスケジューリング専用LSI、ハードウェアスケジューラはプロセッサに代わってEDFスケジューリングを常に行うハードウェアである。一般に汎用のLSIに特定の処理をさせるよりも、その処理に特化されたLSIを用いた方が100倍から1000倍のパフォーマンスの向上が期待できるとされている。組込み機器に多用されているシステムLSIはその代表例である。特に近年においてはハードウェア構築環境が安価で手に入ることもあり、従来ソフトウェアでしか成し得なかった作業をシステムLSIを構築して高速に処理させ、システム全体のパフォーマンスを向上させる動きが多々見受けられる。

本稿にて構築するハードウェアスケジューラでは、スケジューリング機能以外にスケジューリング結果の正当性を確認するスケジューリング可能性判定機能や、スケジューリング結果を状況に応じたより適確なものにするためのリアルタイムモニタリング回路、割込みコントローラを同じくハードウェアで構築し、ハードウェアスケジューラと協調して動作させる。以上の回路を用いて、全ての種類のタスクをハードスケジューラが管理するシステムを構築し、EDFスケジューリングにおけるプロセッサへの過負荷やスケジューリング結果の不一致の問題解決を図る。

2 ハードウェアスケジューラの提案

2.1 EDFハードウェアスケジューラ

ハードウェアスケジューラはスケジューリング専用LSIであり、EDFアルゴリズムを用いた動的スケジューリングを従来よりも高速に行う。EDFアルゴリズムとは非周期的に起動されるタスクについてデッドラインの早いものから順に優先度(プライオリティ)を高くつける方式である。このスケジューリング方式は最大遅れ時間を最小化する最適化スケジュール法である。スケジューリングに必要なパラメータが一つのみであることから、処理の方式が非常に単純なスケジューリング法と言える。そのため応用性が高く、周期的に起動される周期的タスクにも適用が可能である。但し対象とするタスクは実行順序が指定されていない、タスク間に依存関係がない、互いに同期をとらないなどの制約条件がつく。

このハードウェアスケジューラは、従来のOSに実装されているEDFスケジューラにはない以下の特徴がある。

- 高速スケジューリング
  ハードウェアスケジューラはEDFスケジューリングに特化した処理構造を持っており、汎用のプロセッサで行うよりも短いステップ数でスケジューリングを行うことが可能である。従って汎用プロセッサよりもハードウェアスケジューラの

ほうが単位時間に遥かに多くスケジューリングを実行でき、より精度の高いリアルタイムスケジューリングが実現できる。

- プロセッサとの並列動作
  ハードウェアスケジューラはプロセッサと並列動作するので作業効率の向上が望める。通常はスケジューリング、スケジューリングポイントの設定などのタスク管理は通常コンテキスト切り替えの合間か、あるいはタイマを用いて定期的に行われるが、ハードリアルタイム性を求めるならばタスク管理は頻繁に行う必要がある。このタスク管理の少なくないオーバーヘッドはタスクの実行時間に影響する。ハードウェアスケジューラは、プロセッサと並列に動作するためタスク管理を常時行うことができ、プロセッサのタスク管理のオーバーヘッドを軽減できる。またコンテキスト切り替えの回数の減少にもつながり、更に効率のよいリアルタイム処理が達成される。

- 低負荷のタスク管理
  ハードウェアスケジューラの最大の利点はプロセッサ自身がタスクの管理を行わなくてもよいことである。ハードウェアで EDF スケジューリングを実現するため、最適スケジューリングが実現できる上に、プロセッサに負荷がかからずシステムを構築できる。

またハードウェアの高速性を生かし、スケジューリングだけでなくスケジューリング可能性判定を行う。スケジュール可能とは実行可能状態にある全てのタスクがデッドラインを守れるスケジュールが作れることを意味し、このスケジュール可能であるか否かの判定がスケジューリング可能性判定である。スケジューリング可能性判定により、デッドラインを守れないタスクを実行させないことで将棋倒し現象の発生を未然に防ぐ。スケジューリング可能性判定は EDF スケジューリングが実行される度に実行される。

2.2 リアルタイムモニタリング

スケジューラのハードウェア化によるスケジューリングの高速化だけでは、ハードリアルタイム処理は満足に行えない。なぜならスケジューラは与えられたパラメータを用いてスケジューリングをするだけでは、割込みやプロセッサの障害などの予想外の出来事に対処できないからである。よってスケジューラはタスクの性質、プロセッサの実行状況などを考慮しながらスケジューリングする必要があると考える。そこでタスクの実行状況を常に把握できるリアルタイムモニタリング機能をハードウェアスケジューラに追加する。この機能によりスケジューラは現在のプロセッサの実行状況をリアルタイムで把握し、プロセッサの実行状況に応じたスケジューリングを行う。またリアルタイムモニタリング機能を用いると割込み応答時間の予測が可能となる。図 1の例では割込み処理をしていたタスクの詳細がリアルタイムモニタリング機能により明らかになり、タスクがデッドラインを超える前に終了することができる。こうして早期にデッドラインを守れないタスクを発見し、後方に続くタスクへの影響を防ぐ。

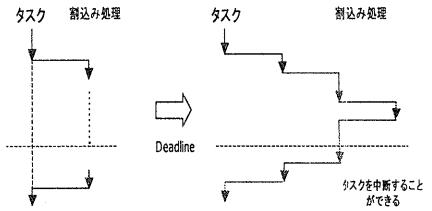

図 1: リアルタイムモニタリングのメリット

この機能を実現するリアルタイムモニタリング回路はプロセッサのプログラムカウンタ、各種リソースを監視する。ハードウェアスケジューラは得られる情報を基にタスクのデッドラインを操作することで状況に応じたタスクの実行を可能とする。

2.3 割込みコントローラ

ハードウェアスケジューラではタスク管理をより正確なものにするため、外部割込みに対する割込みコントローラ機能を持つ。割込みコントローラにより割込み処理も一つのタスク (割込みタスク) として扱い、他のタスクと同様にスケジューリングを行わせることで、ハードウェアスケジューラは割込み応答時間の予測が可能となり、割込みによるスケジューリング結果の不一致を防ぐ。この機能によりハードウェアスケジューラは多重割込みも複数のタスクの起動とみなして対処することができる。

また周期的に起動される周期的タスクの管理機能を付加することで、ハードウェアスケジューラは内部割込み、外部割込み、周期タスクなど全てのタスクを統括し、的確なスケジューリングを可能とする。

3 ハードウェアスケジューラの設計

3.1 システム概要

本システムの概要と処理フローを図2に示す。

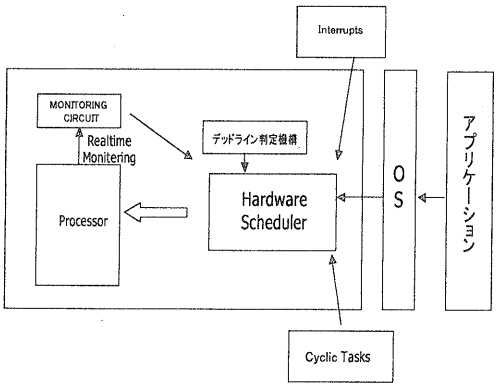

図 2: システムの概要

本スケジューラを用いるスケジューリングの処理フローは以下のとおりである。

1. アプリケーションは、タスクの起動を要求する。
2. タスクの起動要求があるとOSはデッドラインが定まっていないタスクに対しては、予め定めてある相対デッドラインを基にデッドラインを設定する。次にOSはタスクID、実行時間、デッドラインの情報をハードウェアスケジューラに渡す。周期タスクも起動時刻になると同様のタスク情報をハードウェアスケジューラに通知して、周期タスクが起動要求されたことを知らせる。また外部割込みに関しては割込みコントローラが割込み優先度に従ってデッドラインを設定し、タスク情報をスケジューラに通知する。
3. 起動要求された(実行可能状態にある)タスクの情報はReadyQueueと呼ばれる待ち行列に格納される。ハードウェアスケジューラはReadyQueueの中にあるタスク情報をEDFアルゴリズムに従って入れ替えることでスケジューリングを行う。
4. スケジューリングされた順にプロセッサにタスクをディスパッチし、実行させる。
5. リアルタイムモニタリング回路はプロセッサの実行状況を常に監視し、その結果をハードウェアスケジューラに通知する。
6. 通知された情報を元に再スケジューリングを行い、最適なタスクをディスパッチする。

3.2 EDF スケジューラの実装

EDFスケジューラをハードウェアで実現する際に念頭に入れなければならないことは速度と回路規模である。組込み機器に実装することを考慮すると高速にスケジューリングでき、かつ回路規模が小さいものを作成しなければならない。

ハードウェアスケジューラでは用意するReadyQueueの数は一つとする。ReadyQueueを必要な数だけ用意することは回路規模の増加にそのままつながるため、ハードウェア量に制限のある組込み機器では現実的に無理がある。またデッドラインの大小でタスクの実行順序を決めるEDFスケジューラにはプライオリティレベルに応じた複数のReadyQueueを用意しなくてもスケジューリングは行える。

但し一つのReadyQueueではデッドラインの比較に多くの処理ステップがかかる。そこでReadyQueueには整列アルゴリズムの一つであるHeap木を用いて構成する。Heap木とは部分二分木の各点において親の値が左右いずれの子の値よりも大きいデータ構造である(図3参照)。この構造を用いることで親子の大小関係のみで効率よく最大値を導くReadyQueueを構築することができる。このHeap木を複数個用意すれば、更に高速に探索することが可能であるので、後にハードウェアにて拡張を図ることができる。

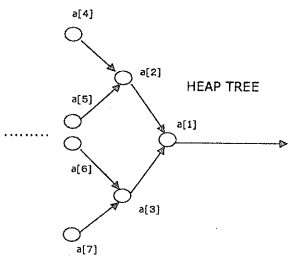

図 3: HEAP 木を用いたReadyQueueの構成

またプロセッサがReadyQueue内のタスクの実行をする順番は、先にReadyQueueの中に入ったものから順に実行するFIFO形式をとる。FIFO形式はEDFアルゴリズムにて周期的タスクを扱うことができ、スケジューリング可能性判定も行えるため、ハードリアルリアルタイムシステムを構築するのには適した方法といえる。

3.3 スケジューリング可能性判定機構

ハードウェアスケジューラ中に実装するスケジューリング可能性判定機構は、スケジューリングされた順番どおりにタスクを実行させたとして、デッドラインを超えるタスクが存在するかを確認する。ここでタスク i の実行時間 C_i としたとき、順当にタスクが実行されたとすればその終了予定時刻は $\sum_{k=1}^{i} C_k$ となる。よって全てのタスクにおいて、この終了予定時刻がタスク i のデッドライン d_i を超えてなければ、スケジュール可能であることが言える。ハードウェアスケジューラは ReadyQueue 内のタスクに変動がある度にこの終了予定時刻を計算し、デッドラインとの比較を行う。

但し Heap 木はその構造上、ReadyQueue の先頭に入っている (木の根にある) タスクしか把握できず、先頭のタスクの終了後にプロセッサが実行するタスクの順番は確定していないためスケジューリング可能性判定は出来ない。そこで Heap 木の先頭に順番が確定しているタスクが入る ReadyQueue(D-Queue) を用意し、この中に入っているタスクを対象にスケジュール可能性判定を行う (図 4参照)。ここでスケジューリング可能性判定が行われるのはヒープ木の先頭のタスクが D-Queue 内に入ってきたとき、つまり D-Queue 内の先頭のタスクが実行されたときである。この構造により高速、小規模、信頼性の全てを得ることのできるスケジューラを構成できる。

なおここで挙げている実行時間とはプリエンプトやブロックされず、純粋にそのタスクの処理に必要な時間を指す。実際の環境では実行時間は確率的に変動する。

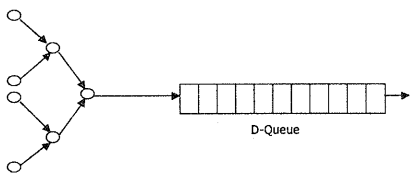

図 4: スケジューリング可能性判定機構の構成

3.4 リアルタイムモニタリング回路の構成

リアルタイムモニタリング回路は現段階ではプロセッサが実行している命令 (プログラムカウンタ) を監視する機能を備えている。このリアルタイムモニタリング回路を用いてハードウェアスケジューラはタスクの終了示すシンボルを検出すると、ReadyQueue の先頭にあるタスクをタスク ID の形でプロセッサに渡す。また実行しているタスクのデッドラインを超過したに場合は強制的にタスクの実行を終了させる。タスク実行終了のシンボルとしては UNIX における exit システムコールを実現する命令群を想定している。

またタスク中にシステムコールの呼び出しがあった場合、ハードウェアスケジューラは呼び出しを感知し、システムコールに対してスケジューリング可能か調べる。スケジューリング可能であれば、実行するシステムコールのタスク ID をプロセッサに通知してシステムコールを実行させる。システムコールの呼び出しに関する従来の方法との比較について図 5に示す。

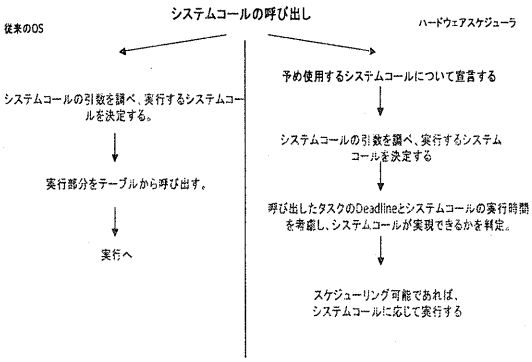

図 5: システムコール時の動作

3.5 割込みコントローラ

ハードウェアスケジューラでは割込み信号の有無を的確に把握するために独自の割込みコントローラを持つ。割込みコントローラの機能は送信された割込み信号を全てマスクせずに受理し、内蔵されている割込みフラグレジスタにチェックを入れる。そしてチェックされている割込みのうち、最も優先度の高い割込みから順にデッドラインを設定する。

割込みタスクのデッドラインを設定する際は、既にスケジューリングしてある ReadyQueue 内のタスクのデッドラインに注意しなければならない。なぜならデッドラインの守れるタスクが、突然入ってきた割込みタスクの影響でデッドラインを超える可能性があるからである。そこでハードウェアスケジューラはスケジューリングされたタスクに影響が及ばないと判断できるタスクとタスクの合間に割込み処理を行わせる。ここでタスク A のデッドラインを d_A、実行時間を C_A、割込みタスクのデッドラインを d_i とすると $d_B > d_A + C_B + d_i$

の条件を満たすタスクA、Bの合間に割込み処理を行わせる。このとき割込みタスクのデッドラインを$d_B - C_B$に設定する。この比較を行うことによって、割込み処理が後続のタスクに影響を及ぼすことを防ぐ。

また割込みコントローラには割込みタスクID、実行時間が記されている割込みタスクテーブルが用意されている。求めたデッドラインと割込みタスクID、実行時間の情報をハードウェアスケジューラに通知し、これらの情報を基にハードウェアスケジューラはEDFスケジューリングを行う。

3.6 周期タスク

タイマなどの周期的に起動されるタスク(以下周期タスク)についても割込みと同様にデッドラインを設定し、タスクID、実行時間、デッドラインをハードウェアスケジューラに通知してEDFスケジューリングを行う。また周期タスクの相対デッドラインは周期と同じとし、そのデッドラインは周期タスクの次の起動時間と設定する。

周期タスクも外部割込みと同様、タスクID、実行時間、周期が示されている周期タスクテーブルをハードウェアで保持しておく。

3.7 オペレーティングシステム

ハードウェアスケジューラとプロセッサとのデータのやり取りはOSを介して行われる．システムに用いられるOSはカーネル機能がそれぞれ独立して存在しているマイクロカーネルOSである。OSに要求されるハードウェアスケジューラに対する機能はハードウェアスケジューラ内のReadyQueueへのタスクの登録機能、及びハードウェアスケジューラからのタスクIDの受信機能である。またメモリ上のタスクの開始番地を記したテーブルは、外部割込み、内部割込み、周期タスクを含めてOSが用意する。

またこのOSはハードウェアスケジューラからタスクIDを通知されると、強制的にディスパッチャを呼び出し、コンテキスト切り替えを行う強制的コンテキスト切り替え機能を持つ。

4 ハードウェアスケジューラの実装

以上の機能を持つハードウェアスケジューラはHDL(ハードウェア記述言語)の一つであるVHDLにて記述し、FPGA(Field-Programable-Gate-Array)を用いて構成している。FPGA上での本システムの詳細図を図6に示す。

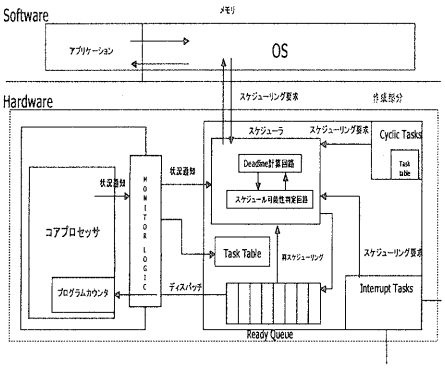

図 6: 全体構造

4.1 データモデル

今回のシステムの実装ではプロセッサからハードウェアスケジューラへの入出力は次のようにする。

- 入力
  タスクID(8bit)、タスクのデッドライン(16bit)、タスク実行時間(8bit)
- 出力
  タスクID(8bit)

ハードウェアスケジューラへの入力は最小限タスク管理に必要な情報である。タスクの起動時刻はEDFアルゴリズムでは規定する必要はない。通常はタスクが終了した時刻が次のタスクの起動時刻となる。これらの情報はアプリケーションの起動に応じてOSや割込みコントローラ、タイマから通知される。一方ハードウェアスケジューラはタスクの実行終了をリアルタイムモニタリング回路を通じて確認すると、ReadyQueue内のスケジューリングされたタスクをタスクIDの形でプロセッサに出力する。但し緊急を要するタスクの場合は現在実行中のタスクをプリエンプトしてタスクIDを出力する。この際、送出するタスクにはレジスタ退避などのコンテキスト切り替え処理を含む。

4.2 追加機能

今回実装するシステムは初期モデルとしてその動作を正確に確認するために、ハードウェアスケジューラだけでなく、対象とするプロセッサも同じくVHDLで

記述し、ハードウェアスケジューラと組み合わせてオンチップで動作させる。実装するプロセッサは 16bit RISC タイプのプロセッサで、ハードウェアスケジューラは作成したプロセッサと同クロックで作業を行う。オンチップによる実装形式は信号線の遅延を極力抑えることができるので複雑な遅延操作が必要なく、ハードウェアスケジューラの動作を容易に検証できる。

更に今回は実装の初期段階として、ハードウェアのみでのシステムの構築を図るためにプロセッサとハードウェアスケジューラをカスタマイズする。具体的にはタスク終了の通知を行う命令 (END 命令) をプロセッサの命令セットに追加し、OS のディスパッチャ機能をハードウェアスケジューラに追加する。

以上の付加機能によりハードウェアのみでタスクの切り替えを行うシステムを構築し、その動作の確認を行う。

4.3 本システムの処理フロー

タスクの実行が終了した状況下で、FPGA 上に実装したハードウェアスケジューラとプロセッサとのデータフローは以下のようになる。

1. プロセッサはタスクの実行終了を END 命令を実行することで通知する。
2. END 命令をモニタリング回路を通じてハードウェアスケジューラは認識する。
3. ハードウェアスケジューラはタスク ID とタスクテーブルからタスクの開始番地を引き出す。
4. 必要に応じてプロセッサのレジスタをハードウェアスケジューラ内のレジスタに退避する。
5. タスクの開始番地をプロセッサ内のプログラムカウンタにセットする。
6. プロセッサはタスクを実行、ハードウェアスケジューラはスケジューリング、及びスケジューリング可能性判定を行う。

5 評価

5.1 実装結果

第4章の仕様からハードウェアスケジューラを VHDL で記述し、FPGA 上に実装を行う。ハードウェアスケジューラに実装したオブジェクトの種類、及びリソース数について表 1に示す。使用したボードは三菱マイコン機器ソフトウェア株式会社製 PowerMedusa(MEB200-A250)、搭載されている FPGA デバイスは Altera 社製 FLEX10KE130VGC599-3 である。また使用した開発環境は Altera 社製 Max+plus2 ver10.0 である。

表 1: ハードウェアスケジューラの構成

タスクの種類	32
タスク数 (HeapTree)	32
タスク数 (SecureQueue)	16
外部割込み信号線	5
周期タスク	8

表 1に示したリソースを有するハードウェアスケジューラを FPGA に実装する。FPGA 内の構成要素はハードウェアスケジューラ (A)、16bit プロセッサ (B)、ハードウェアスケジューラ+プロセッサ (C) とし、それぞれについて FPGA における回路規模 (LCs:Logic Cells)、シミュレーションによる遅延の評価 (frequency) を MAX+plus2 を用いて算出する。その結果を表 2に示す。

表 2: ハードウェアスケジューラの構成

	LCs	frequency(MHz)
(A)	2455	10.25
(B)	1784	23.25
(C)	3849	8.99

5.2 汎用プロセッサとの比較

作成したハードウェアスケジューラの性能を比較するため、スケジューリングにかかるステップ数について汎用プロセッサとハードウェアスケジューラとの比較を行った。比較対象となる汎用プロセッサは 8086 プロセッサとし、この命令セットを用いて EDF アルゴリズムを実行したときの命令数を数えて、それぞれの処理ステップ数を比較する。この際、クロック数は同一とし、処理フローや構造はハードスケジューラと同一条件とする。また取り扱うタスクは既に実行可能状態にあり、整列されているものとする。このタスク数を 4、8、16、32 として、新たにタスクが実行可能状態になったとき、スケジューリングが完了するまでにかかるステップ数の比較した結果を表 3に示す。

表 3の結果から、ハードウェアスケジューラは 8086 プロセッサより短いステップでスケジューリングでき、タスク数が 32 のときハードウェアスケジューラのステップ数は 8086 プロセッサの$\frac{1}{9}$に減少することがわ

表 3: 動作ステップ数の比較

タスク数	HardwareScheduler	Intel 8086
4	5 steps	30 steps
8	6 steps	44 steps
16	7 steps	58 steps
32	8 steps	72 steps

かる。これはハードウェアスケジューラの内部にスケジューリングに必要なパラメータを保持する専用レジスタが備わっているため、主記憶とレジスタ間のデータ移動が伴わないことが原因である。

また表 3にてタスク数の増加に伴い、ハードウェアスケジューラのステップ数はそれほど増加しないことから、ハードウェアスケジューラはタスク数の増加に対して頑健にスケジューリングできるとわかる。

6 検討

本研究においてハードウェアスケジューラの機能の一つであるリアルタイムモニタリングは、専用の回路を設けプロセッサと同一のチップに組み込むことで実現している。システムにリアルタイムモニタリング回路を付加する際はデータ送受信の遅延を極力小さくしなければならないため、オンチップにて実装されていることが望ましい。また同一のチップに組込む場合もカスタマイズによって信号遅延がデータの送受信に影響する可能性があるため、回路中の同期機構やクリティカルパスには注意してカスタマイズする必要がある。

こうした問題に直面しないよう、最近システム LSI 製作で実用されている IEEE1149.1(JTAG) 規格に代表されるオンチップデバッグ回路を利用するリアルタイムモニタリングを検討する。オンチップデバッグ回路にはプロセッサの実行 (プログラムカウンタ) をリアルタイムでトレースできるモードが用意されているため、リアルタイムモニタリング機能が実現できると考えられる。今後はオンチップデバッグ回路を効果的に用いたスケジューリング法について検討する。

7 おわりに

本システムでは過負荷だった EDF アルゴリズムを専用に行うハードウェアを用いて高速にスケジューリングするハードウェアスケジューラを作成した。更にそのハードウェアスケジューラとリアルタイムモニタリング回路を組み合わせて、CPU の実行状況に応じたスケジューリングを行うアーキテクチャを提案した。その結果、タスクの実行時間がデッドラインを超えることを防ぐシステムを構築できた。またハードウェアスケジューラは汎用プロセッサよりも短いステップでスケジューリングを行えることが確認できた、

今後の課題としてはハードウェアスケジューラ用の OS を開発し、実際の環境化での成果を検討する。またハードウェアスケジューラは VHDL で記述されているため、IP(Intelectual Propaty) として配布や再利用されることをを考慮して、様々なシステムにてハードウェアスケジューラを使用できるような構造の検討を行う。

更にビヘイビアレベルで記述されているハードウェアスケジューラに対して更なるゲート数の減少、周波数の向上を図るべく記述方法の工夫をしなければならない。

参考文献

[1] 佐々木 敬泰, 弘中 哲夫: “スケジューリング支援ハードウェアを混載したマルチプロセッサ・システム,” 電気・情報関連学会中国支部第 50 回連合大会, 102010, pp.276–277, 1999.

[2] W.Horn: “Some simple scheduling algorithms,” Naval Research Logistics Quarterly, 21, 1974.

[3] K. M. Zuberi: “Real-Time Operating System Services for Networked Embedded Systems,” PhD thesis, University of Michigan, EECS Dept, 1998.

[4] 飯田 全広, 久我 守弘, 末松 敏則: “マルチスレッド制御ライブラリのハードウェア化によるオンチップ・マルチプロセッサの構成,” Joint Symposium on Parallel Processing' 97, pp.337–344, 1997.

[5] 仲野 巧, 小松 平良, 塩見 彰睦, 今井 正治: “リアルタイム OS チップの性能評価と分散 OS への拡張,” 信学技報, CPSY98-51, pp.23–30,1998.

[6] C.K.Kuy, J.W.Layland, “Scheduling algorithms for multi-programming in a hard-real-time enviroment,” Journal of the ACM, vol. 20, no. 1, pp. 46–61, January 1973.

[7] 村山 仁郎: “8086/80286 プログラマーズハンドブック,” 技術評論社.